MeshKit: An Open-Source Toolkit for Mesh Generation

Tim Tautges, Argonne National Laboratory

Jason Kraftcheck, Univ of Wisconsin-Madison

Jim Porter, Univ of Wisconsin-Madison

The Fathom team:

Argonne: *Alvaro Caceres, Iulian Grindeanu, Rajeev Jain, Dmitry Karpeev, Hong-Jun Kim*

UW: *Shengyong Cai, Steve Jackson, Jiangtau Hu, Brandon Smith, Chaman Verma, Stuart Slattery, Paul Wilson*

SIAM Computational Science & Engineering, Mar 3, 2011

Outline

- Motivation
- The Meshing Problem(s)
- A DiGraph-Based Approach
- MeshKit Design
- Examples

Motivation

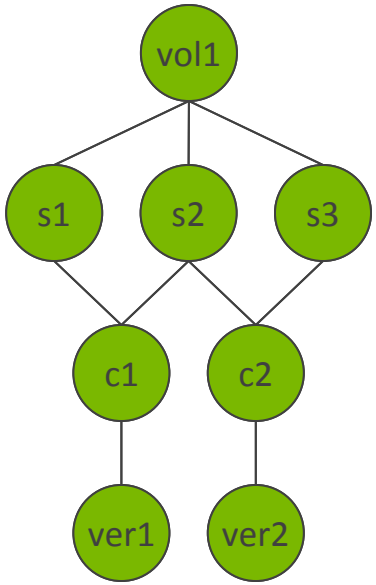
- How do we view the forest (as opposed to the trees) of mesh generation?
  - Traditionally, the meshing process follows geometric topology
  - If you look further, though, it's a bit more general than that, more akin to a digraph-based model (of which a geometric topology is a special case)
- Other things are pretty important when you start looking into the trees, too
  - Infrastructure, especially for geometry-based meshing
  - Availability of a stable of basic meshing capabilities (edge/tri/tet meshers, smoothing, refinement, visualization, quality analysis, …)
- Developing a meshing library is not a new idea (Netgen, gmsh, tetgen, CAMAL, …)
  - However, writing a meshing library that supports interoperability* is
    - *Interoperability in mesh database, geometric model, best-in-class tools and algorithms from various sources
- We need a meshing library that supports needs of two communities
  - Users, so they can generate meshes
  - Developers, so they can start developing specific algorithms right away, instead of after they spend a few years on a good infrastructure
- Open source model more important in meshing than in other areas
  - No matter how good the tool, you'll always reach a point where it doesn't work for your problem
  - Sometimes it's a matter of one key capability that's special to your problem, with the other 95% of your problem treated by existing tools

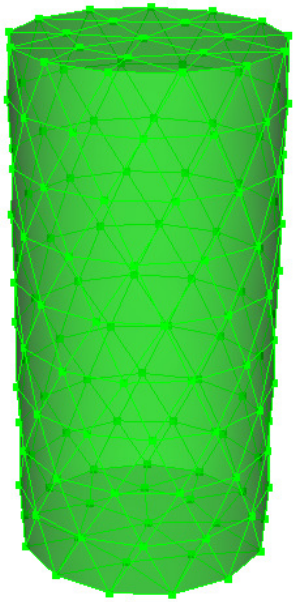
Other Meshing Libraries

- CAMAL 5.1 (SNL, proprietary)
  - Algorithms for tri, quad, tet, hex (sweep, map, submap)
  - Geometric evaluation by application-provided implementations of abstract classes CMLCurveEval and CMLSurfEval
  - API in terms of node positions & connectivity, no high-level “mesh” datatypes
    - No support for assembling bounding mesh from geometry
- Gmsh (GPL)
  - Algorithms for tri, quad (recombine), tet, hex (extrude)
  - Reconstruction of parametric space for disk and non-disk surfaces, using harmonic maps
  - Direct links to OCC-based geometry
- VTK, SciRUN, Salome (LGPL or better)
  - Frameworks that include meshing, along with lots of other stuff
  - Tend to be all or nothing
- In general, difficult to mix and match algorithms from different sources
  - Especially when considering geometric model and use of higher-level API for mesh

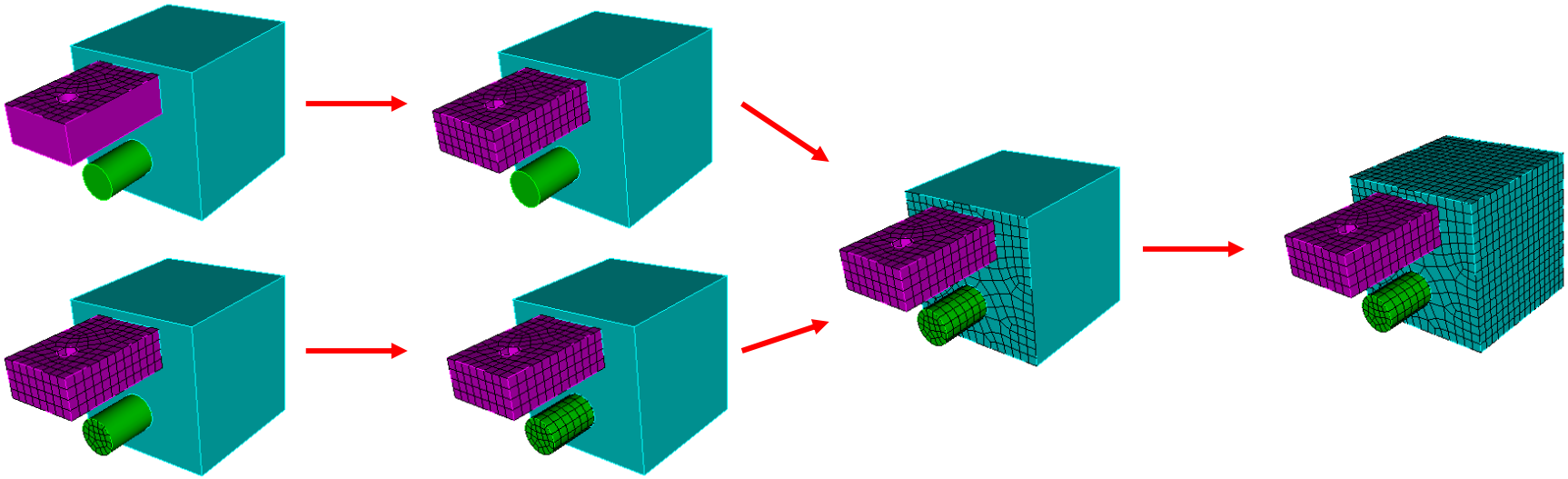
The Meshing Problem(s)

- Geometry-based meshing
  - Mesh vertices
  - Mesh edges
  - Mesh surfaces
  - Mesh volume



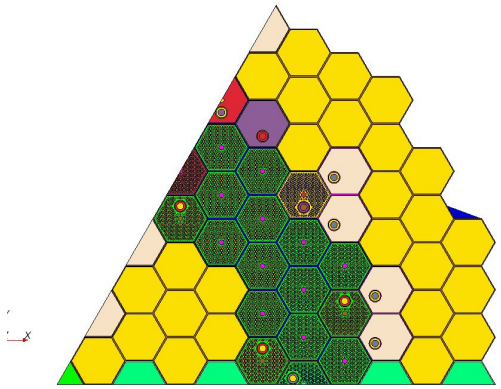


- Sweep dependencies

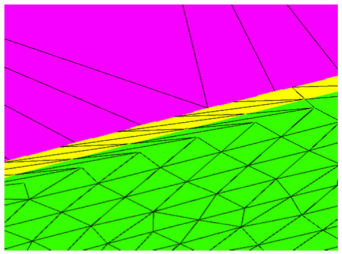


The Meshing Problem(s)

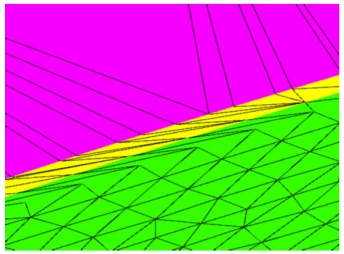
- Mesh Copy/Move/Merge



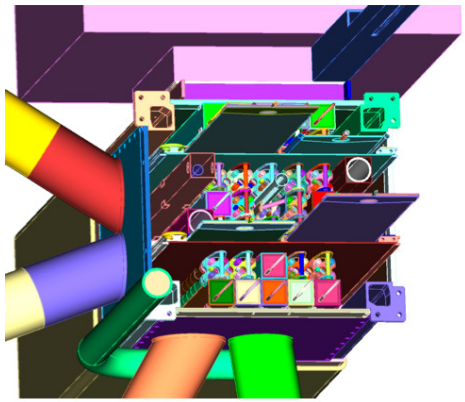
- Watertight models



Unsealed

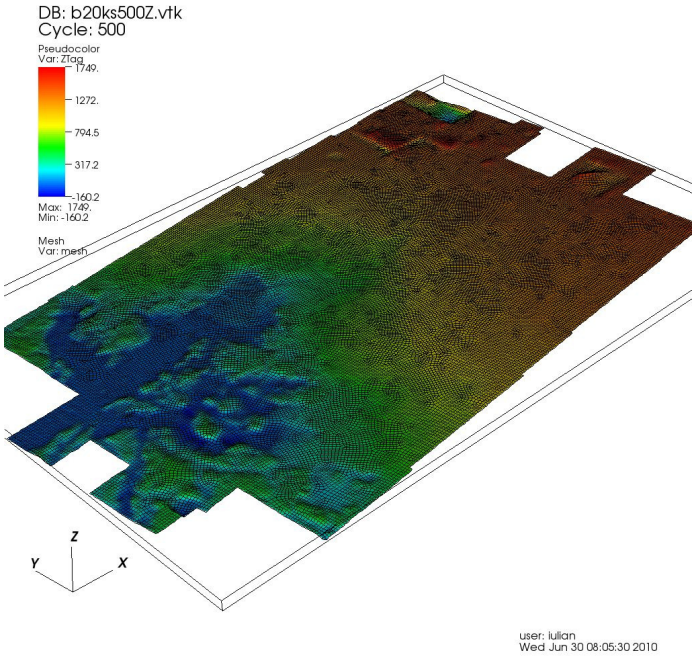


Sealed



UW Nuclear Reactor

- Mesh-Based Geometry



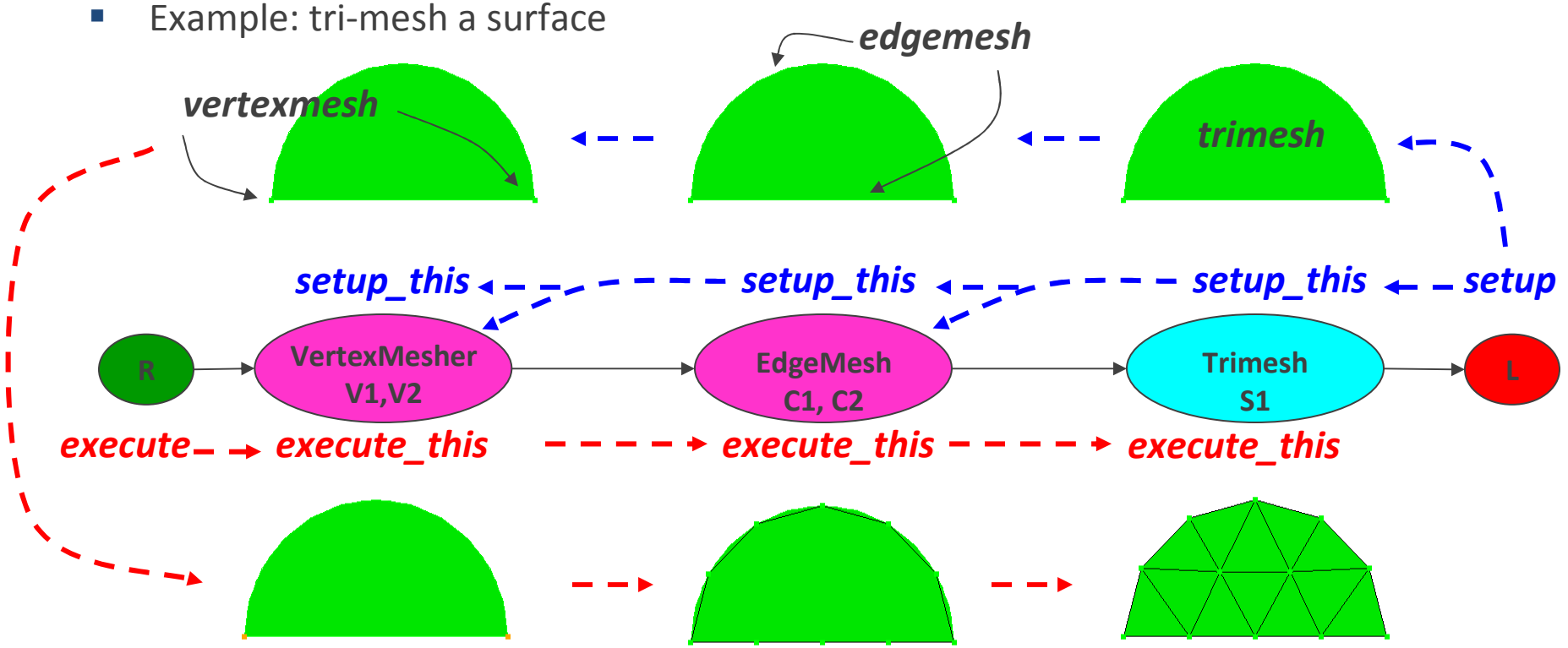
- Embedded Boundary Meshing

DiGraph-Based Organization of Meshing Problem

- Overall, mesh generation can be approached as digraph-based process, with nodes as operations and edges as dependencies
  - More flexible, can express wider variety of complex, mesh-related operations
  - Explicit representation of meshing dependencies allows parallelization of the process
  - May facilitate updating the mesh after small changes
- Various types of graph nodes
  - Meshing algorithm applied to one geometric entity
  - One algorithm applied to a collection of entities
    - VertexMesher, EdgeMesher
  - Mesh-based operation applied to results of preceding operation
    - Mesh, then smooth, then refine
- However, be careful about graph complexity seen by users
  - These graphs can get complicated, fast
  - Support automatic construction of graph nodes, based on constraints from specific meshing algorithms
    - E.g. tet mesher needs surfaces meshed with tris
- 2 phases of graph execution
  - Setup, where algorithms express their requirements (possibly by constructing more nodes)
  - Execute, where mesh gets generated or operation gets performed

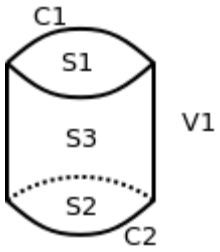
Example of DiGraph-Based Mesh Generation

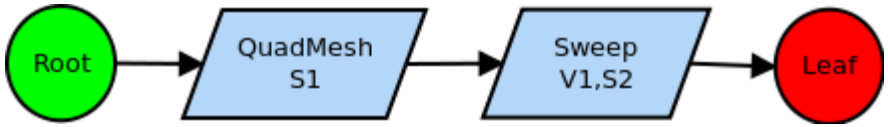
- Directed graph, with single (trivial) root, leaf nodes
- Executed in two phases:
  - Setup: Topological-sort traversal from leaf to root, possibly adding (or moving) nodes upstream
  - Execute: Topological-sort traversal from root to leaf
- Example: tri-mesh a surface

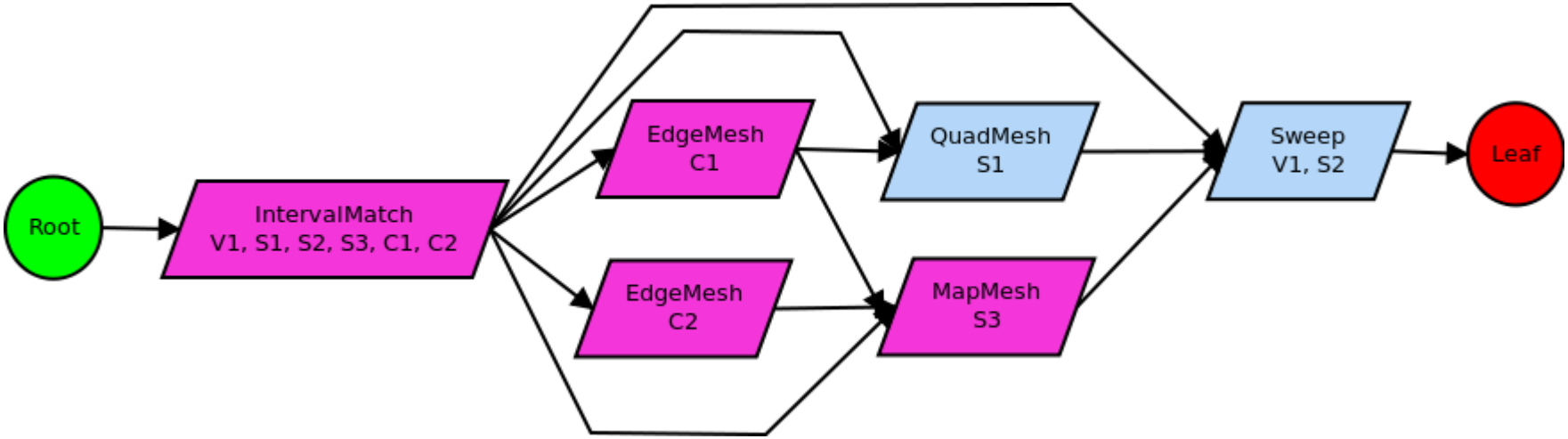


R Root node Automatic User-specified L Leaf node

More Complicated Example: Sweep-Mesh a Volume







MeshKit: C++ Library for Mesh Generation
Core Classes

- MKCore
  - MeshKit instance class
  - isA digraph, hasA (single) root, leaf nodes of digraph
  - Keeps references to geometry, mesh, relations interface instances
- ModelEnt
  - Geometric model entity, mesh set handles
  - Limited set of commonly needed functions, e.g. geometry evaluation, topology traversal
  - In traditional meshing, corresponds to (CAD) geometry entity
    - In other cases, e.g. mesh copy/move, no corresponding CAD entity
  - MeshedState: NO_MESH, BOUNDARY_MESH, SOME_MESH, COMPLETE_MESH, REFINED_MESH, POST_MESH
- MeshOp
  - isA digraph node
  - Keeps a map<ModelEnt*, moab::Range>, to store mesh generated or operated on by this MeshOp
  - Can mesh multiple entities with one MeshOp instance
    - Fewer graph nodes
  - Or, separate instances of a given MeshOp
    - E.g. if different input needed

Dynamic Registration of MeshOps

- MeshKit is meant to support *both* mesh generation users, and mesh generation developers
  - Need to support dynamic registration of new MeshOp classes
- Would like to enable both requesting MeshOps by name, or automated construction based on ability to mesh a given ModelEnt or produce a given mesh entity type
- Accomplish this by requiring a specific set of functions for a MeshOp, and registering the MeshOp with MKCore
- These MeshOp functions need to be defined:
  - Constructor with arg list (MKCore *, MEntVector &)
  - `static const char *name()` – name by which this MeshOp is requested
  - `static const moab::EntityType* output_types()` – array of mesh entity types produced / operated on by this MeshOp (terminated by moab::MBMAXTYPE)
  - `static bool can_mesh(iBase_EntityType dimension)` – returns true if the MeshOp can mesh model entities of this dimension
  - `bool can_mesh(ModelEnt* ent)` – returns true if this MeshOp can mesh the *specified* entity

MeshOp Registration, Construction

- Tried really hard for static registration from each MeshOp's compilation unit
- But, for static library case, if the new MeshOp isn't referenced by name directly or indirectly from an application, it doesn't get pulled in from library at link time
  - Instead, calls to register MK-native MeshOps are put in a few static functions, which are referenced from MKCore
  - Application-defined MeshOps can be registered from main, or from any function linked into final application
- To register a MeshOp: create a global proxy object using:

```
#include "meshkit/RegisterMeshOp.hpp"
RegisterMeshOp<MyMesher> MyMesher_GLOBAL_PROXY;
```

  - Proxy object registered with MeshOpSet singleton accessed through MKCore
  - This object should persist throughout program execution
- Once registered, a new MeshOp can be constructed by name:

```
mk->get_entities_by_dimension(1, curves);
MyMesher *mm = (MyMesher*) mk->construct_meshop("MyMesher", curves);
```

MeshOp setup_this(), execute_this() functions

- setup_this()
  - Creates graph nodes that represent prerequisites, e.g. for surface mesher, mesh the boundary; that is, it "sets up" the meshing process for entity(ies)
  - For trivial or automatic cases, can call MeshOp::setup_boundary()
    - Automatic: use default scheme for dimension, which is 1st MeshOp type for that dimension registered in MKCore
  - This function often results in new MeshOps getting created, and put in the graph upstream of the current MeshOp
  - For more complicated situations, e.g. automatic scheme selection, this may involve deeper changes to the graph
    - Needs more study
- execute_this()
  - Is where the typical mesh generation or mesh-based operation happens
  - If setup was done properly, all prerequisites will be satisfied by the time this is called
  - Most put generated mesh into MeshOp::MEntSelection (map<ModelEnt*, moab::Range>)
    - Provides *some* reversibility
    - At this point, don't intend to store extra data, e.g. new vertex positions, so not completely reversible
- MKCore, MeshOp both implement setup(), execute(), which just perform the setup and execute traversals from leaf/root (MKCore) or current node (MeshOp)

Example MeshOp: EdgeMesher

- setup_this():

```
for (MEntSelection::iterator mit = mentSelection.begin();
     mit != mentSelection.end(); mit++) {
    ModelEnt *me = mit->first;
      // check if already meshed, and return if so
    if (me->get_meshed_state() >= COMPLETE_MESH ||
        me->mesh_intervals() > 0) continue;
     // check/compute the intervals for this ModelEnt
    check_intervals(me);
}
   // assign bounding vertices to VertexMesher
setup_boundary();
```

- execute_this():
  - Typical parametric-space edge meshing
  - Equal, bias, dual (bias), and curvature options

Example MeshOp: CAMALPaver

- (see src/extern/CAMAL/CAMALPaver.cpp)

External Libraries

- Important to support *both* OSS *and* non-open libraries/algorithms
- General rule: if library/algorithm is LGPL or compatible, will bundle that code with MeshKit
  - Netgen (tri, tet)
  - Qslim1.0 (decimation)
  - Mesquite (smoothing)
  - Verdict (mesh quality)
  - LEMON (graph)
- Otherwise, support through wrappers enabled by configure-time option
  - CAMAL (SNL)
  - Gmsh (GPL)
  - Other

Other Details

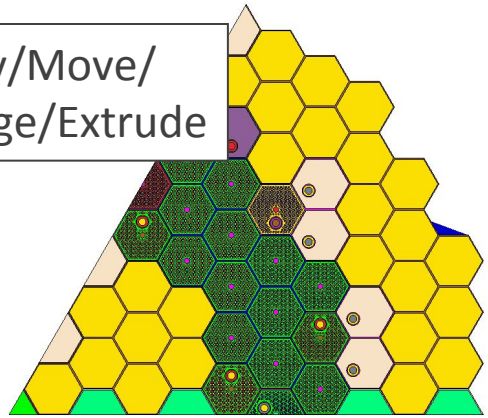
- Graph functionality imported from Lemon graph library
  - Like Boost Graph Library, but with fewer templates-on-templates
  - Right now, accessing graph through mix of methods on MeshKit classes and direct calling of Lemon functions on graph + nodes/edges returned from those classes
    - Probably better to make a GraphNode class in MeshKit core, put all graph-accessing functions and members there
  - Right now, only MeshOps can be graph nodes
    - Eventually, probably need others, e.g. maybe ModelEnt should also be derived from GraphNode
- That pesky interface question
  - Geometry and relations accessed through iGeom, iRel
  - For mesh, access through MOAB or iMesh interfaces
    - Need MOAB for trees, ranges, Skinner, other MOAB-only capabilities
    - iMesh and MOAB instances both available from MKCore
    - To support other databases, will require implementing MOAB interface on top of iMesh or the other database

Other Details (2)

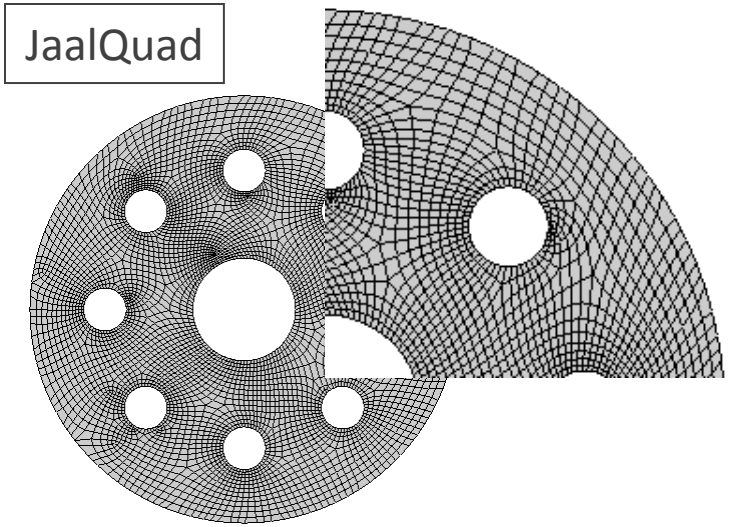
- Scripting
  - Eventually, will have Python interfaces generated automatically
  - It's been recommended to me that we also bundle a Python interpreter directly with MeshKit; sounds like a good idea, at least for some apps
- Documentation extremely important
  - Everything in MeshKit should be born (and probably conceived) documented
  - Will use doxygen almost exclusively for documentation
    - It can do more than most people realize

Algorithms Available in MeshKit 0.9…

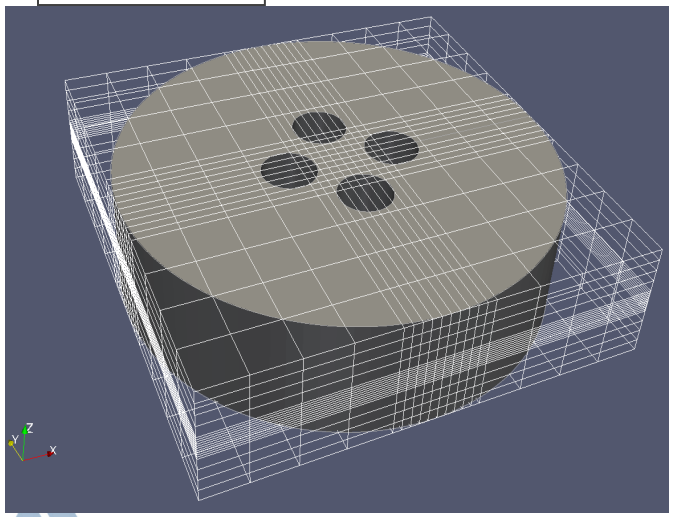
Copy/Move/
Merge/Extrude



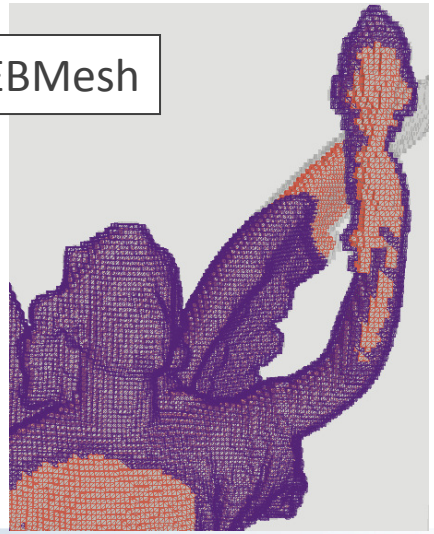
JaalQuad



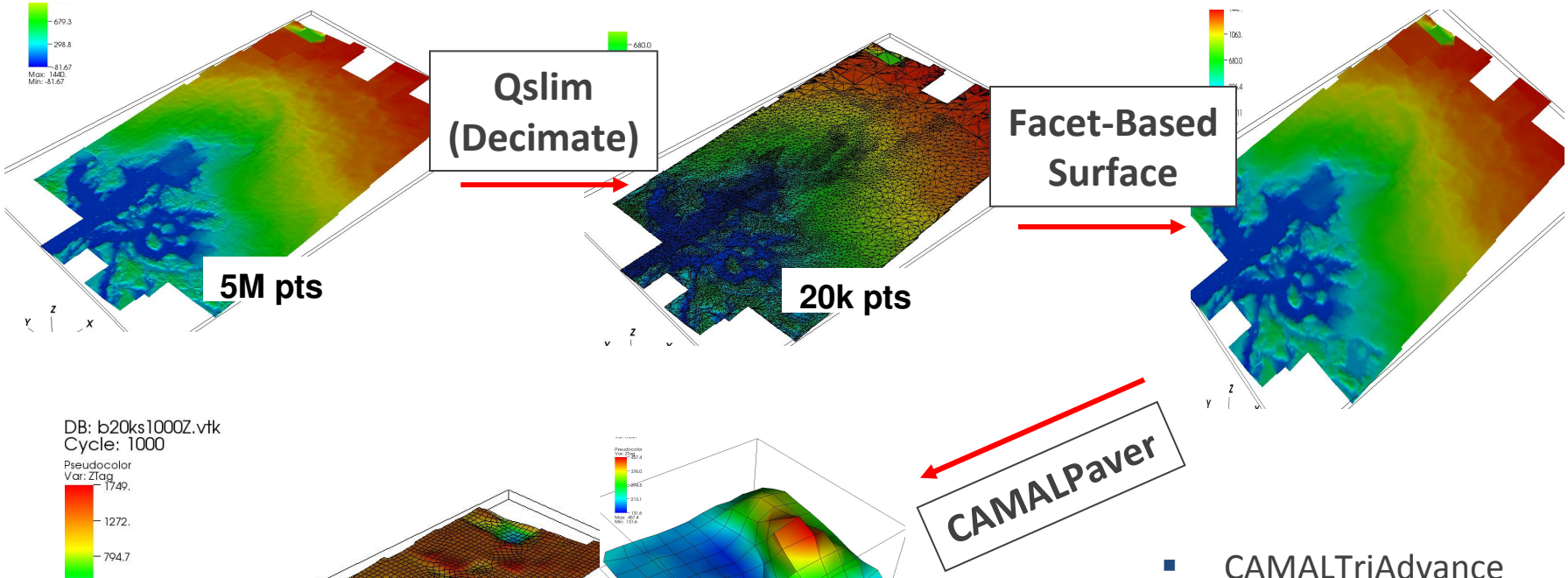
SCDMesh



EBMesh



Algorithms Available in MeshKit 0.9…



- CAMALTriAdvance
- CAMALTetMesher
- NGTetMesher (Netgen)
- OneToOneSwept
- MesquiteSmooth
- EdgeMesher
- VertexMesher
- (others)

Future Plans

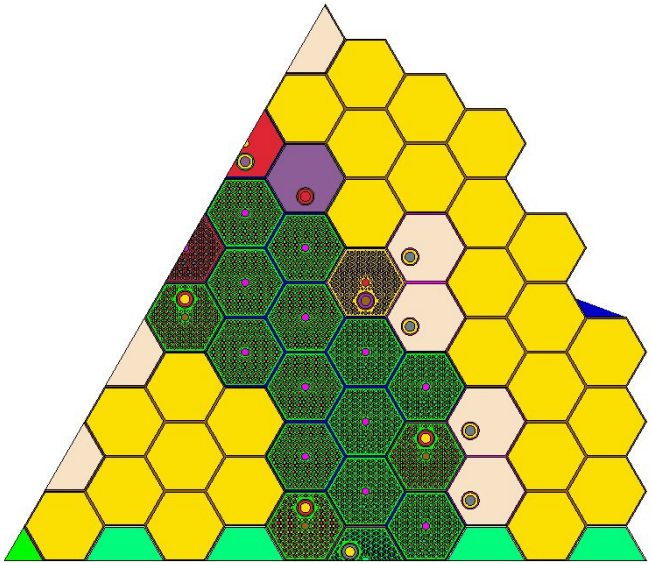
- Native tri mesher (Triangle? Netgen? Home-grown?)
- Tri, tet, hex (parallel) refinement
- Parallel mesh merge
- Auto-generated Python interfaces
- Interval matching
- More work on graph-based approach

Conclusions

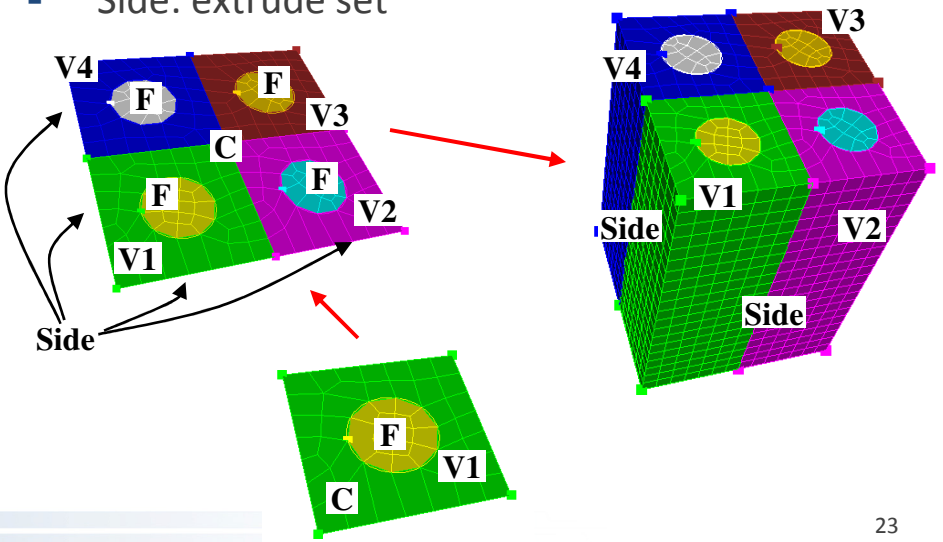
- Need an open-source mesh generation environment, that
  - Supports both users and developers
  - Has bridges to non-open-source meshing algorithms
  - Is flexible in how various algorithms interact
- Graph-based meshing process captures many of the relevant workflows in mesh generation
- MeshKit designed to support these uses
- V0.9 (almost) ready (targeting mid-April release)
- Meshkit-announce (https://lists.mcs.anl.gov/mailman/listinfo/meshkit-announce) for details

Copy/Move/Merge/Extrude

- CopyMesh, ExtrudeMesh, MergeMesh
- Works with any type of 2D, 3D elements
- 15mins start to finish for 12M element hex mesh (Linux desktop workstation)
  - Geometry construction
  - Assembly meshing
  - Copy/move/merge

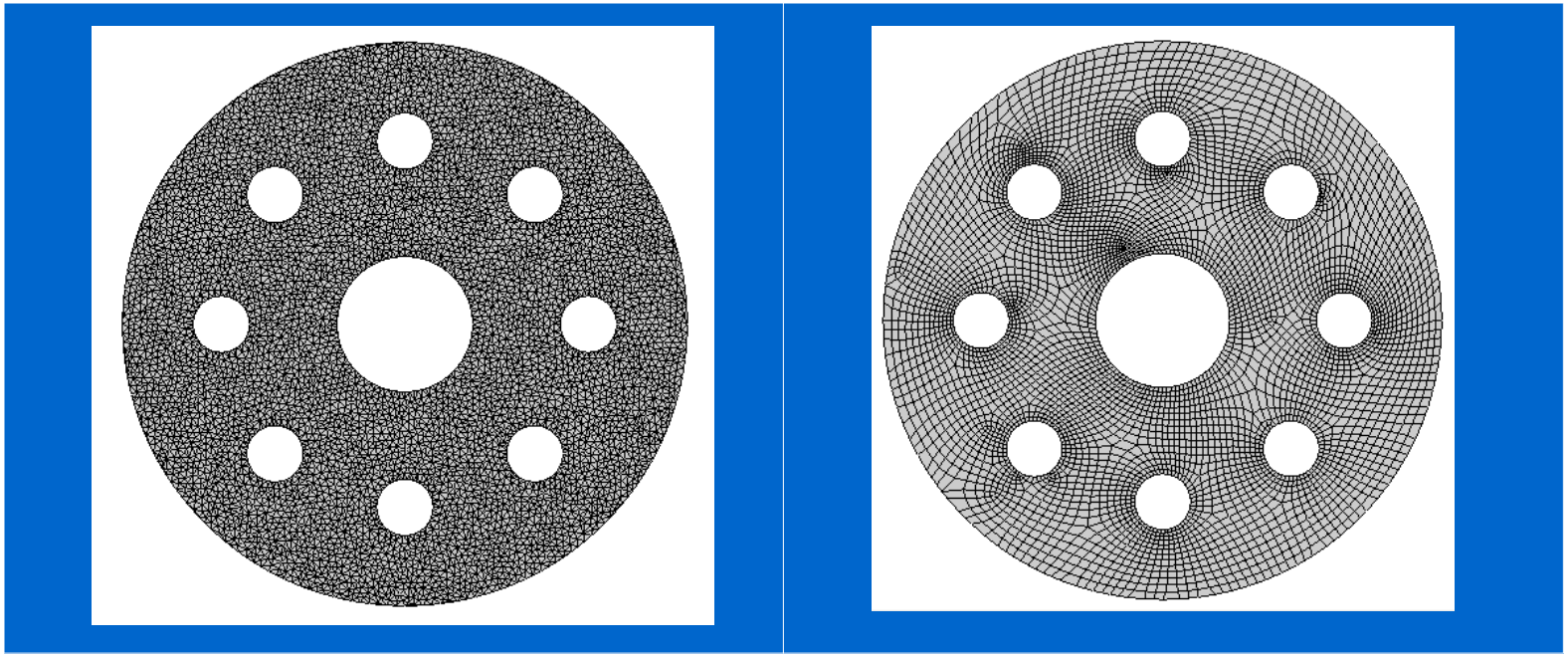


- Boundary conditions, material definitions handled using set abstractions:
  - Copy set: new set with entity copies
  - Expand set: entity copies added to same set
  - Extrude set: entities replaced with (d+1)-dimensional extrusions
- Allows handling of various set types with a common abstraction
- Will work similarly for refinement
- F, C: expand sets
- V1: copy set
- Side: extrude set



Jaal QuadMesher

- Tri-quad conversion using deterministic tree matching algorithm
- Cleanup: local (singlet, doublet, diamond removal), global (Bunin's algorithm)
- Provable robustness, with very few irregular nodes in final mesh
- Currently, 20k quadrilaterals/sec (~2.4GHz Linux workstation)

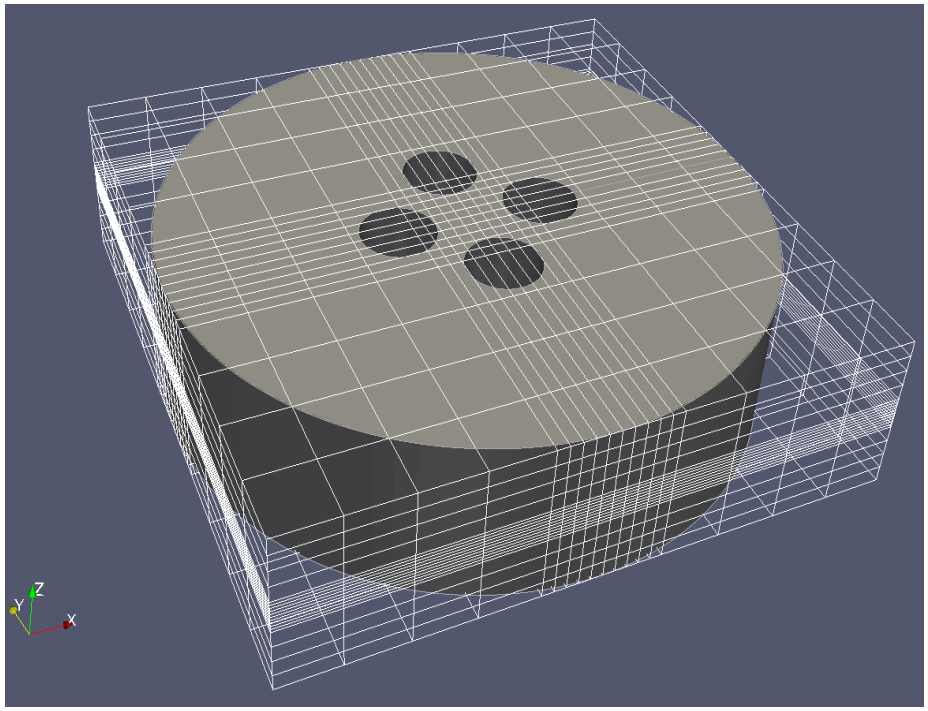


5658 nodes, 10669 triangles

5192 nodes, 4868 quads

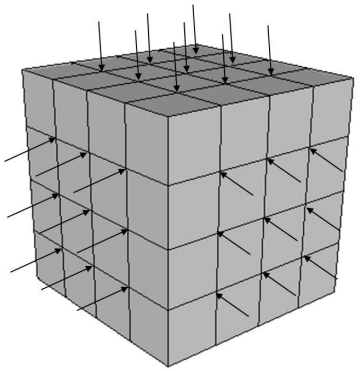
Structured Background Mesher (SCDMesh)

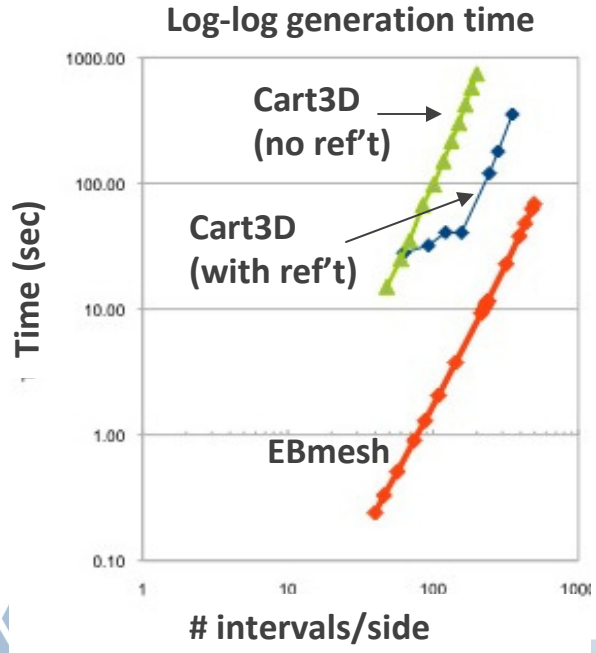
- Generates a structured Cartesian grid using a geometric entity's bounding box
- Adjustable or equal spacing
- Supports EBMesh, and basic structured mesh generation

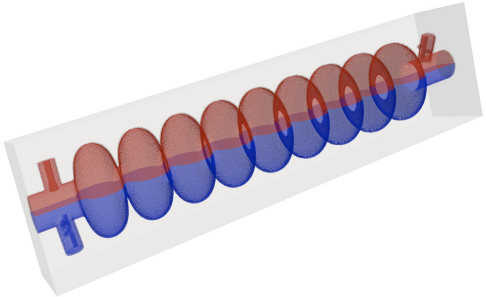


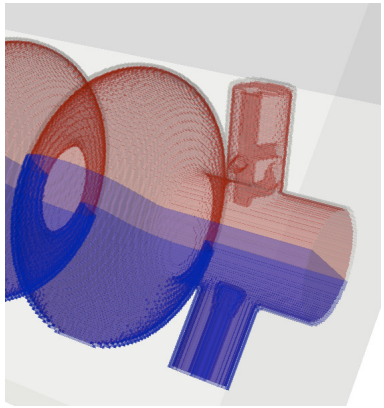
Embedded Boundary Mesh Generation (EBMesh)

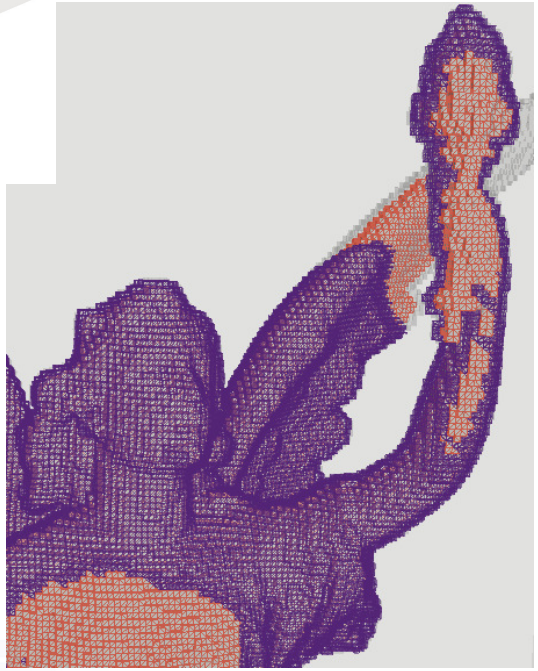
- Find inside/boundary/outside elements in Cartesian mesh surrounding a body
- Uses fast, robust ray tracing based on hierarchical OBBs in MOAB
- 10x faster than Cart3D
- Uses SCDMesh





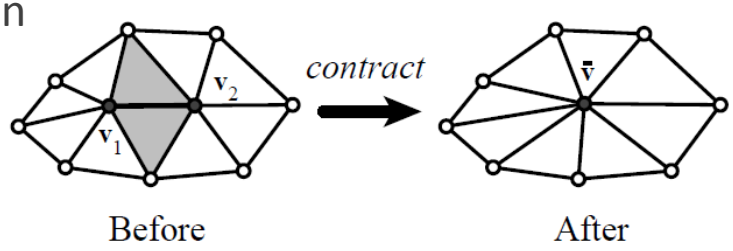




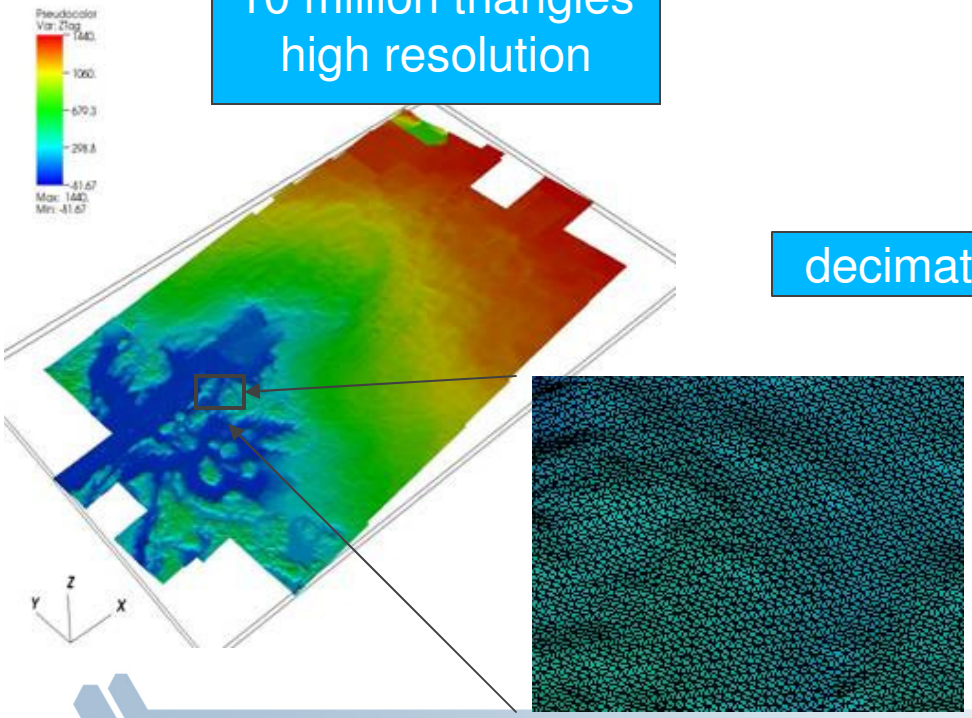


Decimation (Qslim)

- Surface mesh decimation based on Qslim1.0 (Garland, UIUC)
- Main method of decimation: Edge Contraction
- Minimize certain error (quadrics)

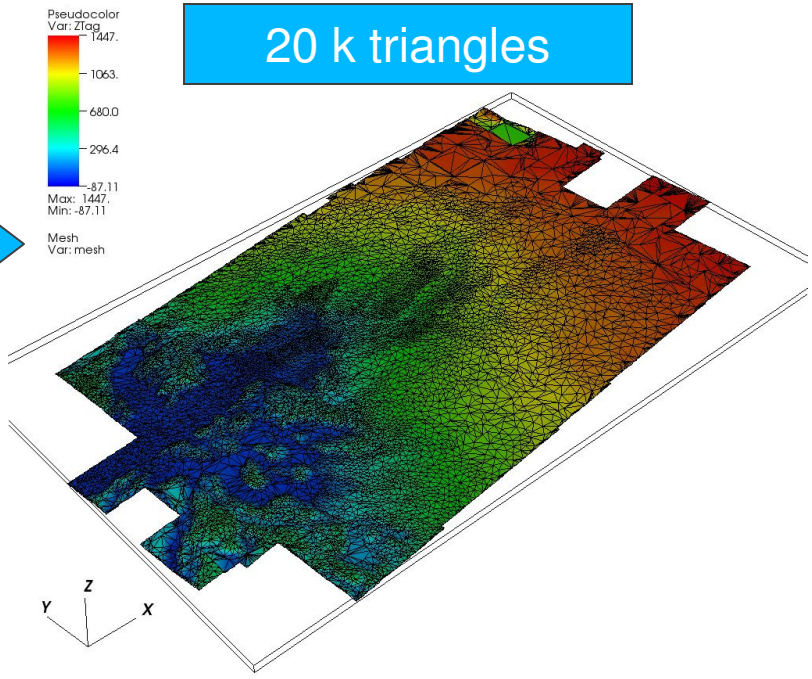


10 million triangles high resolution

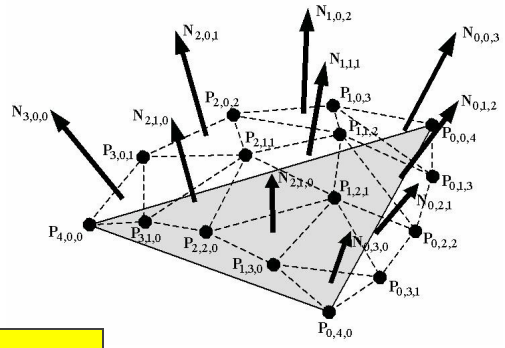


decimation

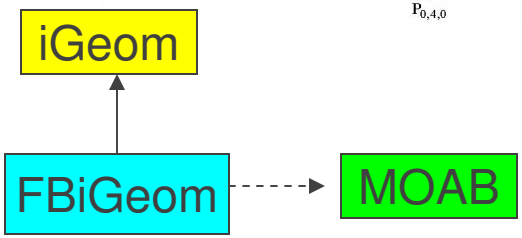
20 k triangles

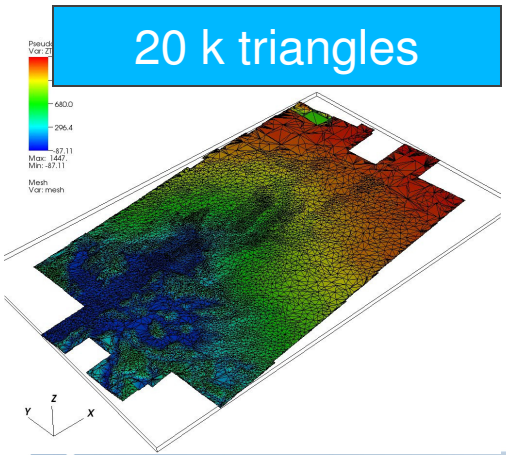


Facet-Based Geometry (FBiGeom)



- C1-continuous facet-based geometric representation (Owen et al, '02), implemented in MOAB
- FBiGeom provides limited iGeom API based on smooth facet-based geometry
- 2 options: linear or smooth evaluations





CAMALPaver